

Improving k-Means Clustering Performance with Disentangled Internal Representations

Abien Fred Agarap
College of Computer Studies
De La Salle University
Manila, Philippines
abien_agarap@dlsu.edu.ph

Arnulfo P. Azcarraga
College of Computer Studies
De La Salle University
Manila, Philippines
arnulfo.azcarraga@dlsu.edu.ph

Abstract—Deep clustering algorithms combine representation learning and clustering by jointly optimizing a clustering loss and a non-clustering loss. In such methods, a deep neural network is used for representation learning together with a clustering network. Instead of following this framework to improve clustering performance, we propose a simpler approach of optimizing the *entanglement* of the learned latent code representation of an autoencoder. We define *entanglement* as how close pairs of points from the same class or structure are, relative to pairs of points from different classes or structures. To measure the entanglement of data points, we use the *soft nearest neighbor loss*, and expand it by introducing an annealing temperature factor. Using our proposed approach, the test clustering accuracy was 96.2% on the MNIST dataset, 85.6% on the Fashion-MNIST dataset, and 79.2% on the EMNIST Balanced dataset, outperforming our baseline models.

Index Terms—clustering, disentanglement, encoding, internal representations

I. INTRODUCTION AND RELATED WORKS

Clustering is an unsupervised learning task that groups a set of objects in a way that the objects in a group share more similarities among them than those from other groups. It is a widely-studied task as its applications include but are not limited to its use in data analysis and visualization, anomaly detection, sequence analysis, and natural language processing. Like other machine learning methods, clustering algorithms heavily rely on the choice of feature representation. For this reason, the design of preprocessing pipelines and feature transformations (a.k.a. *feature engineering*) consumes a considerable amount of time. In turn, the task of feature engineering plays a crucial role in the success of machine learning methods. However, due to its labor-intensive nature, it hinders faster development, autonomous learning, and reusability across tasks.

We take k -means clustering as an example, which typically uses the Euclidean distance among points in a given feature space (e.g. for images, it could be the raw pixels or gradient-orientation histograms); for difficult image datasets like CIFAR10 [1], clustering with Euclidean distance on raw pixels may be ineffective.

Hence, the move towards automatically learning the best representation for a given data has gained mainstream attention since the success of deep learning for computer vision [2]–[4],

where the exceptional gains on benchmark tasks have resulted from automatically learning better feature representation. The task of automatically learning feature representation is known as *representation learning*.

More explicitly, representation learning is the task of learning the most salient features of a given data, i.e. features that imply the underlying structure of the data. It is implicitly done in a supervised deep neural network by using its hidden layers to learn and to provide representations for its last layer, thereby rendering a task such as classification or regression easier. For instance, data points that are not linearly separable in the raw feature space may become linearly separable at the last hidden layer through the composition of feature representations in the hidden layers. So, by automatically learning the representations instead of feature engineering, we are unencumbered of the task to learn the best possible feature representation for better performance in downstream tasks such as classification, clustering, and regression. To further take advantage of representation learning, it may be explicitly designed to forge representations in favor of a downstream task such as clustering. In the following subsections, we briefly discuss related works that use the aforementioned strategy.

A. Deep Embedded Clustering (DEC)

Xie et al. (2016) [5] introduced the Deep Embedded Clustering (DEC), a method that simultaneously learns feature representations and cluster assignments using a deep neural network. DEC learns a mapping $f_\theta : X \rightarrow Z$ where X is the original feature space, while Z is the lower-dimensional feature space. Then, it iteratively optimizes the Kullback-Leibler divergence to minimize the within-cluster distance of each cluster in Z . They had a clustering accuracy of 84.30% on the MNIST dataset [6].

B. Variational Deep Embedding (VaDE)

Another approach to strongly influence learned representations to favor clustering is the Variational Deep Embedding (VaDE) by Jiang et al. (2016) [7]. It is an unsupervised generative clustering approach that employs the framework of Variational Autoencoder [8], by combining a Gaussian Mixture Model (GMM) and a deep neural network (DNN). Specifically, a cluster is chosen by the GMM from which the

latent representation z is sampled, and then the DNN decodes z to an observable data x . Their approach had a clustering accuracy of 94.46% on the MNIST dataset [6].

C. ClusterGAN

Similar to DEC [5] and VaDE [7], ClusterGAN [9] also uses learned representation for clustering, but their approach built on the Generative Adversarial Network (GAN) [10] framework. That is, they incorporated a clustering-specific loss term to the minimax objective. They had a clustering accuracy of 95% on the MNIST dataset [6] and 63% on the Fashion-MNIST dataset [11].

D. N2D: (not too) deep clustering

DEC [5], VaDE [7], and ClusterGAN [9] are all deep clustering algorithms, i.e. methods that combine a deep neural network as a data encoder $f_\theta : X \rightarrow Z$, and a clustering network, that jointly learns a clustering loss and a non-clustering loss (e.g. minimax objective for GAN). In contrast, McConville et al. (2019) [12] proposed N2D (Not too Deep) clustering, wherein they performed manifold learning on the latent code representation from an autoencoder, and used the learned manifold for clustering. They found that using UMAP in their framework achieves the best clustering-friendly manifold of the latent code representation. Using this approach, they had a clustering accuracy of 94.8% on the MNIST dataset [6] and 67.2% on the Fashion-MNIST dataset [11].

Similar to N2D [12], we propose a relatively simpler approach compared to deep clustering algorithms. But we also use an autoencoder to learn the latent code representation of a data, then use the said representation for clustering. We draw our difference on how to learn a more clustering-friendly latent code representation.

Instead of learning such a representation by using a manifold learning technique such as Isomap [13], t-SNE [14], or UMAP [15], we regularize the autoencoder reconstruction loss with the *soft nearest neighbor loss* [16], [17]. The said loss function measures the lack of separation of class (or structural, for unsupervised tasks) manifolds in representation space—in other words, the *entanglement* of data points from different classes or structures.

We focus on minimizing the entanglement of class manifolds in a latent code representation to derive a more clustering-friendly representation.

Our contributions are as follows:

- 1) We expand the soft nearest neighbor loss by introducing an annealing temperature factor, and we use it to learn a latent code representation that is primed for clustering (Section II).
- 2) We present comparatively strong clustering results in terms of clustering accuracy, normalized mutual information, and adjusted Rand index (Section III).
- 3) A simple yet effective way of clustering on (disentangled) latent code representation (Section III).

II. LEARNING DISENTANGLED REPRESENTATIONS

We consider the problem of clustering a set of N points $\{x_i \in X\}_{i=1}^N$ into k clusters, each represented by a centroid $\mu_{j \in 1, \dots, k}$. Instead of directly clustering the original features X , we transform the data with a non-linear mapping $Z = enc(X)$, where Z is the latent code representation. But to learn a more clustering-friendly representation, we propose to learn to *disentangle* them, i.e. isolate class- or structure-similar data points, which implicitly maximizes the inter-cluster variance.

A. Autoencoder

An autoencoder is a neural network that aims to find the function mapping the features x to itself through the use of an encoder function $h = enc(x)$ that learns the latent code representation of the features, and a decoder function that reconstructs the features from the latent code representation $r = dec(h)$. To learn the reconstruction task, it minimizes a loss function $\mathcal{L}(x, dec(enc(x)))$, where \mathcal{L} is a function penalizing the decoder output $dec(enc(x))$ for being dissimilar from x . Typically, this reconstruction loss is the Mean Squared Error (MSE) $\frac{1}{n} \sum_{i=1}^n \|dec(enc(x_i)) - x_i\|_2^2$. Then, similar to other neural networks, it is usually trained with a gradient-based method aided with backpropagation of errors.

The reconstruction task of an autoencoder has a by-product of learning good internal representations, and so, we take advantage of this for clustering, particularly, the latent code representation $z = enc(x)$ – thus drawing our similarity with N2D [12]. For our experiments, we used the binary cross entropy (Eq. 1) as the reconstruction loss.

$$\ell_{rec}(x, r) = \frac{1}{n} \sum_{i=1}^n -x_i \log(r_i) + (1 - x_i) \log(1 - r_i) \quad (1)$$

We used this in lieu of MSE since our features X were normalized to values $[0, 1] \in \mathbb{R}$.

B. Soft Nearest Neighbors Loss

In our context, we define *entanglement* as how close pairs of representations from the same class or structure are, relative to pairs of representations from different classes or structures. Frosst et al. (2019) [17] used the same term in the same context. A low entanglement implies that representations from the same class or structure are closer than they are to representations from different classes or structures. To measure the entanglement of representations, Frosst et al. (2019) [17] expanded the non-linear neighborhood component analysis (NCA) [16] objective by introducing the temperature factor T , and called this modified objective the *soft nearest neighbor loss*.

They defined the *soft nearest neighbor loss* as the non-linear NCA at temperature T , for a batch of b samples (x, y) ,

$$\ell_{sn}(x, y, T) = -\frac{1}{b} \sum_{i=1 \dots b} \log \left(\frac{\sum_{\substack{j \in 1 \dots b \\ j \neq i \\ y_i = y_j}} e^{-\frac{\|x_i - x_j\|^2}{T}}}{\sum_{\substack{k \in 1 \dots b \\ k \neq i}} e^{-\frac{\|x_i - x_k\|^2}{T}}} \right) \quad (2)$$

where x may be the raw input features or the learned representations in the hidden layers of a neural network. We may describe soft nearest neighbor loss as the negative log probability of sampling a neighboring point j from the same class as i in a batch b , similar to the probabilistic sampling by Goldberger et al. (2005) [18]. A low value of soft nearest neighbor loss is tantamount to a low entanglement (high disentanglement). In our experiments, we used cosine similarity instead of Euclidean distance.

1) *Temperature*: Frosst et al. (2019) [17] described the temperature factor T as a way to control the relative importance given to the distances between pairs of points, i.e. at low temperatures, the loss is dominated by small distances while the actual distances between widely separated representations become less relevant. Conversely, at high temperatures, the distances between widely separated points dominate the loss.

2) *Annealing Temperature*: We build on this idea of the temperature influencing the importance of distances between pairs of points, and extend the soft nearest neighbor loss by introducing an annealing temperature (Eq. 3) instead of a fixed one,

$$T = \frac{1}{(\eta + i)^\gamma} \quad (3)$$

where i is the current training epoch, and we set $\eta = 1$ and $\gamma = 0.55$ for our experiments, similar to Neelakantan et al. (2015) [19]. We show a disentangled latent code representation learned with the soft nearest neighbor loss, both with fixed temperature and with annealing temperature, in Figure 1. In both cases, as we minimize the soft nearest neighbor loss, the latent code representation becomes more clustering-friendly. However, with annealing temperature, we gain a lower entanglement at an earlier training epoch, thus the disentanglement of representations starts earlier than with a fixed temperature.

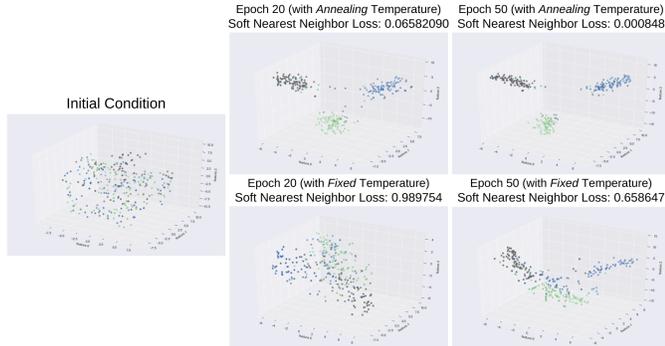


Fig. 1. Comparing the soft nearest neighbor loss with annealing temperature and with fixed temperature. We sampled and randomly labelled 300 data points from a Gaussian distribution, and ran gradient descent on them with soft nearest neighbor loss. The figure at the left shows the initial condition of the labelled points. We can see the separation of clusters in the latent code from epoch 20 to epoch 50, rendering the classes more isolated. We present disentangled representations on benchmark datasets later in the paper. This figure is best viewed in color.

We also used the lowest soft nearest neighbor loss among the hidden layers of our autoencoder:

$$\ell'_{sn} = \arg \min \ell_{sn}(x, y, T) \quad (4)$$

Using the arg min configuration, we achieved a more stable computation of the soft nearest neighbor loss during training.

Now that we have defined our contribution, we lay down the objective function used by our autoencoder to learn the disentangled representations for clustering. We define a composite loss (Eq. 5) that consists of the autoencoder reconstruction loss ℓ_{rec} , and the soft nearest neighbor loss ℓ_{sn} , with an α parameter which directly influences the disentanglement computation – we set $\alpha = 100$. Note that our goal is not necessarily to learn a good reconstruction of the input data, but to learn a good disentangled representation that we can use to improve clustering performance.

$$\mathcal{L}(f, x, y) = \ell_{rec}(x, r) + \alpha \cdot \sum_{i \in k} \ell_{sn}(f^i(x), y) \quad (5)$$

Another contribution of this work is the simplicity of our proposed method. So, unlike DEC [5], VaDE [7], and ClusterGAN [9] which uses an auxiliary clustering network, we use a simple k-Means clustering [20] on the disentangled latent code representations.

Thus far, we have discussed our proposed method of disentangling learned representations to improve clustering performance. However, autoencoding and clustering are both unsupervised learning tasks, while we are proposing to use the soft nearest neighbor loss, a loss function that uses labels to illuminate the class similarity structure of internal representations learned by a neural network. With this in mind, we formulated two different soft nearest neighbor loss functions: (1) supervised, and (2) unsupervised. In the unsupervised setting, we simply perform the same probabilistic sampling of a neighboring point j to i , but we do not constrain them to come from the same class.

To simulate the lack of labelled data, we used a small labelled subset of the benchmark datasets for the supervised configuration of the soft nearest neighbor loss. The different soft nearest neighbor loss configurations we used in our experiments are listed in Table I.

TABLE I
CONFIGURATIONS OF AUTOENCODER (AE) TRAINED WITH SOFT NEAREST NEIGHBOR LOSS (SNNL)

Shorthand	Full SNNL-trained Autoencoder Configuration
SNNL-1	Supervised SNNL-trained AE w/ fixed T
SNNL-2	Unsupervised SNNL-trained AE w/ fixed T
SNNL-3	Supervised SNNL-trained AE w/ arg min and fixed T
SNNL-4	Unsupervised SNNL-trained AE w/ arg min and fixed T
SNNL-5	Supervised SNNL-trained AE w/ annealing T
SNNL-6	Unsupervised SNNL-trained AE w/ annealing T
SNNL-7	Supervised SNNL-trained AE w/ arg min and annealing T
SNNL-8	Unsupervised SNNL-trained AE w/ arg min and annealing T

We use Table I as a lookup table of shorthand to save space on Tables III, IV, and V. Model configurations labelled as SNNL 1–4 use a fixed temperature, while those that are labelled as SNNL 5–8 use our annealing temperature (Eq. 3). Moreover, model configurations with even numbers use the unsupervised soft nearest neighbor loss, while the ones with odd numbers use the supervised version.

Finally, we summarize the details of our proposed method as follows,

- 1) Train an autoencoder with the composite loss (Eq. 5) using a gradient-based method with backpropagation.
- 2) Use the disentangled latent code representation $z = enc(x)$ of the autoencoder for k-Means clustering.

III. CLUSTERING ON DISENTANGLED REPRESENTATIONS

To demonstrate the effectiveness of our approach, we conducted experiments on benchmark datasets, and lay down the clustering performance of the related models on the same benchmark datasets we used.

A. Autoencoder Model

We used a fully connected network with $d - 500 - 500 - 2000 - c - 2000 - 500 - 500 - \hat{d}$ units, where d and \hat{d} (s.t. $d = \hat{d}$) refer to the dimensionality of the features, and c refers to the dimensionality of the latent code representation – similar to the one used by Salakhutdinov and Hinton (2007) [16]. The c -latent code layer and \hat{d} -reconstruction layer used logistic function while the remaining hidden layers used ReLU function [22]. All hidden layers were initialized with He initialization [21]. We set $c = 70$ across all datasets and models for a more fair comparison. Finally, we trained our model using Adam [23] with a learning rate of 1×10^{-3} for 50 epochs on all our datasets.

B. k-Means Clustering

We ran k-Means clustering, with centroids initialized using `k-means++` [24], on the disentangled latent code representation from our autoencoder for nine times. The first run started with 10 iterations, with each run incremented by 10, i.e. the first clustering ran for 10 iterations, while the ninth ran for 90 iterations. We recorded the clustering performance on the ninth run for each model on each dataset.

C. Evaluation Metrics

For all the models, we used the number of ground-truth categories in each dataset as the number of clusters. We used six different metrics to evaluate the clustering performance of our baseline and experimental models. For the first three metrics, the values lie in the interval $[0, 1]$, where values closer to 1 correspond to a better clustering performance. The values for the fourth metric lie in the interval $[-1, 1]$, where values near 1 are better while values near -1 are worse. The last two metrics are unbounded, but only the first of which implies a better clustering performance when its values are higher, while the last metric requires a lower value for better performance.

1) *Clustering Accuracy*: In clustering, accuracy (ACC) is defined as the best match between the ground-truth labels and the predicted clusters [25]. Using the ground-truth labels as pseudo-cluster labels is known as *cluster assumption* in the semi-supervised learning literature [26].

$$ACC = \max_m \frac{\sum_{i=1}^n \mathbb{1}\{l_i = m(c_i)\}}{n}, \quad (6)$$

where l_i is the ground-truth label, c_i is the cluster prediction, and m ranges over all possible one-to-one mappings between clusters and labels.

2) *Normalized Mutual Information*: The Normalized Mutual Information (NMI) is the normalization of mutual information (MI) score to have its value within $[0, 1] \in \mathbb{R}$, where 0 denotes no mutual information while 1 denotes perfect correlation. It is formally defined as follows,

$$NMI = \frac{2I(y, c)}{[H(y) + H(c)]} \quad (7)$$

where y is the ground-truth label, c is the cluster prediction, H is the entropy, and I is the mutual information between the ground-truth labels and the cluster predictions.

3) *Adjusted Rand Index*: The Adjusted Rand Index (ARI) [27] is the Rand Index (RI) adjusted for chance. RI is the similarity between two clusterings by considering all pairs of points and counting pairs assigned to the same or different clusters in the predicted and true clusterings (see Eq. 8).

$$RI = \frac{TP + TN}{TP + FP + FN + TN} \quad (8)$$

where TP is the true positive, TN is the true negative, FP is the false positive, and FN is the false negative. ARI is then computed from RI by using Eq. 9,

$$ARI = \frac{RI - \mathbb{E}[RI]}{\max(RI) - \mathbb{E}[RI]} \quad (9)$$

ARI values lie within $[0, 1] \in \mathbb{R}$, where 0 denotes random labelling independently of the number of clusters, while 1 denotes the clusterings are identical up to a permutation.

4) *Silhouette Score*: The Silhouette Score (SIL) [28] measures the similarity of examples to their own cluster compared to other clusters, and it is computed by using Eq. 10,

$$SIL = \frac{b(i) - a(i)}{\max[a(i), b(i)]} \quad (10)$$

where $a(i)$ is the distance between point i and all other points in its cluster, and can be computed by using Eq. 11,

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j) \quad (11)$$

where C_i is the predicted cluster for point i , and $d(i, j)$ is the distance between points i and j .

Then, $b(i)$ is the distance between point i and all other points in the next nearest cluster, and can be computed by using Eq. 12,

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j) \quad (12)$$

Any distance metric may be used, but we used the Euclidean distance metric in our evaluation.

5) *Calinski-Harabasz Score*: The Calinski-Harabasz score [29] (CHS) is defined as the ratio between within-cluster dispersion and between-cluster dispersion, and it is computed by using Eq. 13. A higher CHS implies better cluster separation.

$$CHS = \frac{Tr(B_k)}{Tr(W_k)} \times \frac{N - k}{k - 1} \quad (13)$$

where B_k is the between-cluster dispersion matrix given by Eq. 14,

$$B_k = \sum_q n_q (c_q - c)(c_q - c)^T \quad (14)$$

and W_k is the within-cluster dispersion given by Eq. 15,

$$W_k = \sum_{q=1}^k \sum_{x \in C_q} (x - c_q)(x - c_q)^T \quad (15)$$

where N is the number of points in a data, C_q is the set of points in cluster q , c_q is the center of cluster q , c is the center of C , and n_q is the number of points in cluster q .

6) *Davies-Bouldin Index*: The Davies-Bouldin Index (DBI) is defined as the ratio of within-cluster distances to between-cluster distances [30], [31]. A lower DBI implies better cluster separation. It is computed by using Eq. 16,

$$DBI = \frac{1}{k} \sum_{i=1}^k \max_{i \neq j} R_j \quad (16)$$

where R is the average similarity among clusters given by Eq. 17.

$$R_{ij} = \frac{s_i + s_j}{d_{ij}} \quad (17)$$

where s_i is the cluster diameter which is the average distance between each point in cluster i and the cluster centroid, and d_{ij} is the distance between centroids i and j .

D. Datasets Description

We evaluate and compare our baseline and experimental methods on three image datasets. We list the dataset statistics in Table II.

TABLE II
DATASETS STATISTICS.

Dataset	# Samples	Input Dimension	# Clusters
MNIST	70,000	784	10
Fashion-MNIST	70,000	784	10
EMNIST Balanced	131,600	784	47

- 1) *MNIST*: The MNIST handwritten digit classification dataset [6] consists of 60,000 training examples and 10,000 test examples – all in grayscale, and of size 28 by 28 pixels. We reshaped each image to a 784-dimensional vector.
- 2) *Fashion-MNIST*: The Fashion-MNIST [11] is a more challenging alternative to the MNIST dataset, which consists of 60,000 training examples and 10,000 test examples – also all in grayscale, and of size 28 by 28

pixels. We reshaped each image to a 784-dimensional vector.

- 3) *EMNIST Balanced*: The EMNIST Balanced is a subset of the EMNIST handwritten characters classification dataset [32], that has a 47 balanced classes of handwritten digits and letters. It has 112,800 training examples and 18,800 test examples – also all in grayscale, and of size 28 by 28 pixels. We reshaped each image to a 784-dimensional vector.

For the supervised configuration of the soft nearest neighbor loss, we simulate the lack of labelled data by randomly picking 10,000 labelled training examples each for the MNIST and the Fashion-MNIST datasets, and randomly picking 20,000 labelled training examples for the EMNIST balanced dataset, since it has to be clustered in 47 groups. Despite using a small subset of labelled training examples, we still used the full test sets for the clustering task.

On the other hand, we used the full training sets for k-Means clustering on the original feature representation, and for the unsupervised learning models (i.e. baseline autoencoder, SNNL- $\{2, 4, 6, 8\}$), and we evaluated them on the full test sets.

E. Clustering Performance

We evaluate the performance of our experimental method in its different configurations. For our baseline models, we used k-Means clustering on (1) the original feature representation, encoded using principal components analysis (PCA) to 70 dimensions in order to avoid the “curse of dimensionality” [33], and (2) on the latent code representation from an autoencoder trained with reconstruction loss (Eq. 1) only. Then, we retrieved the reported clustering performance of DEC [5], VaDE [7], ClusterGAN [9], and N2D [12] from literature as additional baseline results. We report the average clustering performance across four runs of each model, as well as their best clustering performance on each of the benchmark dataset.

We present empirical evidence that consistently shows our method significantly outperforms all our baseline models on each of the benchmark dataset we used. Specifically, our method configurations SNNL-5 and SNNL-7.

1) *MNIST*: We had the highest average clustering accuracy of **95.5%** on the MNIST dataset using SNNL-7 configuration, and the highest best clustering accuracy of **96.2%** using SNNL-5 configuration. Meanwhile, for each of our related models, only their best clustering accuracy was reported. Their results are as follows: DEC had 84.3%, VaDE had 94.5%, N2D had 94.8%, and ClusterGAN had 95%. If we follow the related works to report the best performance, we outperformed our closest baseline, ClusterGAN, with **1.2%** in clustering accuracy. For the full results of clustering performance on the MNIST dataset, we refer the reader to Table III.

2) *Fashion-MNIST*: We had the highest clustering accuracy of **84.4%** (average) and **85.6%** (best) on the Fashion-MNIST dataset using SNNL-5. Meanwhile, our related models had the following results: ClusterGAN had 63% and N2D had 67.2%. However, we should note here that N2D was trained on the

TABLE III
CLUSTERING PERFORMANCE ON THE MNIST DATASET.

Method	ACC		NMI		ARI		SIL		CHS		DBI	
	Average	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average	Best
Original	0.525	0.547	0.497	0.499	0.367	0.367	0.078	0.078	469.231	471.494	2.605	2.592
SNNL-2	0.549	0.552	0.512	0.52	0.387	0.393	0.086	0.089	443.535	459.947	2.631	2.619
SNNL-4	0.562	0.569	0.516	0.521	0.4	0.409	0.086	0.089	442.462	458.644	2.640	2.583
Baseline AE	0.56	0.576	0.518	0.528	0.399	0.419	0.086	0.088	445.146	457.095	2.638	2.613
SNNL-6	0.562	0.58	0.512	0.53	0.4	0.429	0.084	0.088	437.957	449.735	2.647	2.616
SNNL-8	0.575	0.589	0.528	0.543	0.413	0.433	0.085	0.088	437.458	447.399	2.657	2.636
DEC [5]*	–	0.843	–	–	–	–	–	–	–	–	–	–
VaDE [7]*	–	0.945	–	–	–	–	–	–	–	–	–	–
N2D [12]*	–	0.948	–	0.882	–	–	–	–	–	–	–	–
ClusterGAN [9]*	–	0.95	–	0.89	–	0.89	–	–	–	–	–	–
SNNL-1	0.901	0.952	0.862	0.881	0.849	0.898	0.957	0.96	162605.531	214754.189	0.245	0.092
SNNL-3	0.904	0.957	0.860	0.896	0.821	0.908	0.7	0.775	14874.154	24043.605	0.591	0.484
SNNL-7	0.953	0.962	0.891	0.903	0.897	0.918	0.742	0.893	24713.490	44316.457	0.473	0.204
SNNL-5	0.955	0.958	0.890	0.895	0.903	0.911	0.874	0.887	33782.991	39916.318	0.239	0.186

TABLE IV
CLUSTERING PERFORMANCE ON THE FASHION-MNIST DATASET.

Method	ACC		NMI		ARI		SIL		CHS		DBI	
	Average	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average	Best
SNNL-8	0.52	0.548	0.561	0.569	0.385	0.395	0.120	0.125	793.908	824.502	2.143	2.103
SNNL-2	0.535	0.553	0.568	0.576	0.397	0.406	0.118	0.123	798.417	833.797	2.141	2.088
Baseline AE	0.540	0.557	0.567	0.581	0.391	0.413	0.119	0.123	806.924	830.819	2.119	2.098
Original	0.518	0.559	0.518	0.531	0.366	0.389	0.186	0.191	1662.815	1673.995	1.61	1.549
SNNL-6	0.553	0.591	0.569	0.579	0.401	0.434	0.116	0.12	796.24	808.859	2.181	2.119
SNNL-4	0.555	0.595	0.574	0.583	0.408	0.436	0.119	0.122	790.444	803.54	2.167	2.11
ClusterGAN [9]*	–	0.63	–	0.64	–	0.50	–	–	–	–	–	–
N2D [12]*	–	0.672	–	0.684	–	–	–	–	–	–	–	–
SNNL-3	0.672	0.693	0.682	0.691	0.527	0.535	0.575	0.59	10031.276	11013.237	0.666	0.626
SNNL-1	0.741	0.748	0.756	0.763	0.619	0.633	0.963	0.967	212599.963	272354.021	0.331	0.225
SNNL-7	0.832	0.848	0.753	0.765	0.696	0.711	0.665	0.722	10638.715	16001.514	0.692	0.599
SNNL-5	0.844	0.856	0.762	0.767	0.714	0.729	0.672	0.705	9646.826	10823.988	0.628	0.546

entire Fashion-MNIST dataset, i.e. both training and test sets. For the full results of clustering performance on the Fashion-MNIST dataset, we refer the reader to Table IV.

3) *EMNIST Balanced*: We had the highest clustering accuracy of **78.5%** (average) and **79.2%** (best) on the EMNIST Balanced dataset using SNNL-5 configuration, while our baseline autoencoder had 35.6% (average) and 36.1% (best). Neither the deep clustering algorithms nor N2D used the EMNIST Balanced dataset (or any other EMNIST subsets). For the full results of clustering performance on the EMNIST Balanced dataset, we refer the reader to Table V.

The full results in Table III, IV, and V not only show that our experimental methods outperformed our baseline methods, but also show that the supervised configurations (SNNL- $\{3, 5, 7\}$) of the soft nearest neighbor loss performed better than the unsupervised ones (SNNL- $\{2, 4, 6, 8\}$), with the exception of SNNL-1 on the EMNIST Balanced dataset. This emphasizes the requisite of labels for the soft nearest neighbor loss to illuminate the neighborhood structure of a dataset, thus learning a more clustering-friendly feature representation.

Furthermore, SNNL- $\{3, 5, 7\}$ also outperformed our baseline methods in terms of NMI and ARI. Intuitively, having high NMI and ARI scores for clustering implies that there is a high number of similar data points in each cluster. In other

words, similar data points have been correctly assigned to their clusters, which in our case was based on the pseudo-cluster labels we used. We can also see that SNNL- $\{1, 3, 5, 7\}$ had the best scores in terms of SIL, CHS, and DBI. Having a high SIL score implies that the points in a cluster are more similar among themselves than they are to the points from a different cluster. Then, both CHS and DBI measure cluster separation, i.e. CHS defines better separated clusters through variance ratios, while DBI defines better separated clusters through distances among clusters.

Our results support that our method and its variants were able to learn a more clustering-friendly feature representation by having better defined clusters and by having correctly clustered data points, which we visually inspect in the next subsection.

F. Visualizing Disentangled Latent Representation

We show the resulting disentangled latent representation for the test set of MNIST, Fashion-MNIST, and EMNIST Balanced datasets in Figure 2. These latent code representations were obtained after training an autoencoder with SNNL-7 for 50 epochs on a subset of 10,000 training examples each for MNIST and Fashion-MNIST datasets, and on a subset of 20,000 training examples of EMNIST Balanced dataset. However, since the EMNIST Balanced dataset has 47 clusters,

TABLE V
CLUSTERING PERFORMANCE ON THE EMNIST-BALANCED DATASET.

Method	ACC		NMI		ARI		SIL		CHS		DBI	
	Average	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average	Best
Original	0.321	0.33	0.418	0.42	0.173	0.176	0.051	0.052	237.078	237.413	2.659	2.636
SNNL-1	0.319	0.349	0.662	0.674	0.335	0.356	0.897	0.902	71353.661	78749.196	1.079	1.069
SNNL-6	0.348	0.352	0.441	0.444	0.193	0.196	0.041	0.041	222.108	225.193	2.712	2.695
SNNL-2	0.343	0.353	0.439	0.442	0.191	0.197	0.037	0.04	223.75	224.457	2.698	2.658
SNNL-4	0.344	0.353	0.438	0.445	0.190	0.196	0.038	0.041	225.839	227.568	2.674	2.641
SNNL-8	0.35	0.356	0.442	0.444	0.195	0.197	0.04	0.042	223.449	225.474	2.692	2.669
Baseline AE	0.356	0.361	0.446	0.449	0.198	0.201	0.042	0.044	224.537	228.909	2.709	2.687
SNNL-3	0.396	0.44	0.667	0.699	0.391	0.438	0.6	0.737	11138.224	16324.421	1.378	1.345
SNNL-7	0.701	0.743	0.753	0.775	0.529	0.634	0.580	0.761	4832.462	7098.482	0.852	0.647
SNNL-5	0.785	0.792	0.776	0.783	0.641	0.655	0.677	0.687	4697.866	5025.238	0.646	0.607

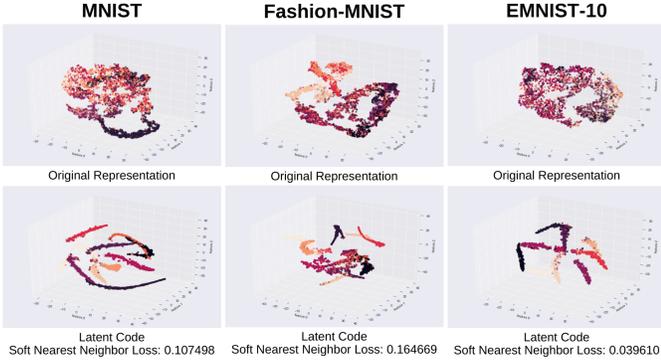


Fig. 2. Three-dimensional visualization comparing the original representation and the disentangled latent representation of the three datasets. To achieve this visualization, the representations were encoded using t-SNE with perplexity = 50 and learning rate = 10, optimized for 5,000 iterations, with the same random seed set for all computations. However, for clustering, we used higher dimensionality to achieve better clustering performance. This figure is best viewed in color.

we only visualized a randomly chosen 10 clusters for easier and cleaner visualization. We can see in the aforementioned figure that the latent code representation for each dataset indeed became more clustering-friendly by having well-defined clusters (indicated by the cluster dispersion) and correct cluster assignments (indicated by the cluster colors).

G. Clustering on Fewer Labelled Examples

In Figure 3, we show that even with fewer labelled training examples, the clustering accuracy on disentangled latent code representation is still better than on the original feature representation and on the latent code representation from a baseline autoencoder. We trained an autoencoder with SNNL-1 and SNNL-5 configurations on randomly picked 1,000 examples, 3,000 examples, and 6,000 examples from the MNIST and Fashion-MNIST datasets, and treated them as labelled data. Then we evaluated on the full test sets. In contrast, we used the full MNIST and Fashion-MNIST sets for both the original feature representation and the latent code representation from our baseline autoencoder.

On MNIST, our best model configuration was SNNL-5 that had a clustering accuracy of 90.85%, 92.95%, and 94.78% when trained on 1,000 examples, 3,000 examples, and 6,000

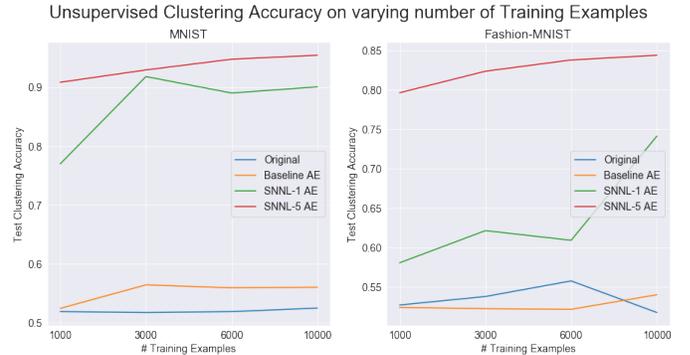


Fig. 3. Test clustering accuracy on the MNIST and Fashion-MNIST test sets when small subsets of labelled data are used for training. Both the original representation and the baseline autoencoder do not take advantage of the labelled dataset.

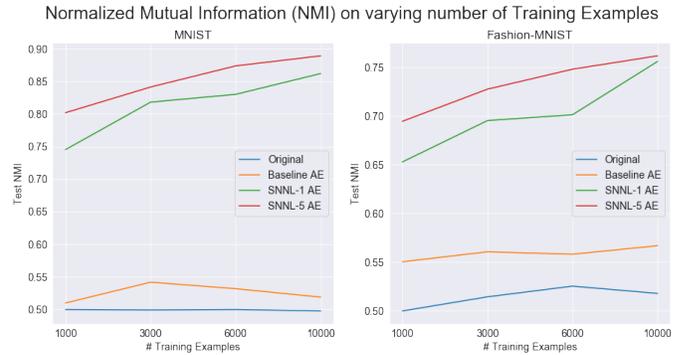


Fig. 4. Normalized mutual information (NMI) on the MNIST and Fashion-MNIST test sets when small subsets of labelled data are used for training. Both the original representation and the baseline autoencoder do not take advantage of the labelled dataset.

examples respectively. Using the same model configuration on Fashion-MNIST, we had a clustering accuracy of 79.63%, 82.35%, and 83.78% when trained on 1,000 examples, 3,000 examples, and 6,000 examples respectively.

The results in Figure 3 are average clustering accuracy across four runs of each model on the small labelled subsets of MNIST and Fashion-MNIST datasets. To support these results, we can observe the same trend of clustering performance in terms of NMI in Figure 4. With this, we draw a parallel with

the non-linear NCA [16] performance, where they employed unsupervised pre-training for kNN classification, and found an even better test error rate when they fine-tuned on a small fraction of labelled MNIST dataset – a test error rate of 1%.

We argue that this robustness of clustering performance, despite smaller labelled subsets were used for SNNL-trained autoencoders, is due to the soft nearest neighbor loss enabling the learning of a latent code representation that takes into account the distances among pairs of points from the same class or structure relative to points from different classes or structures. In other words, it brings out the neighborhood structure of a dataset.

Finally, we were able to further improve the clustering performance on disentangled latent code representation with our annealing temperature factor. The intuition as to how this annealing factor improves the soft nearest neighbor loss builds on the results in Figure 1, and may be described as follows: as the training progresses, the data points of similar class or structure become closer. Hence, there are fewer widely separated points of the same class or structure as training progresses. Consequently, this renders the use of high temperatures less relevant over time, since the class- or structure-similar points are already becoming entangled – which in turn, makes the latent code representation disentangled.

IV. CONCLUSION

Compared to deep clustering methods [5], [7], [9], we employed a simpler approach to cluster the latent code representation from an autoencoder. We used a composite loss of the autoencoder reconstruction loss and the soft nearest neighbor loss to learn a more clustering-friendly latent code representation from an autoencoder, thereby improving the k-Means clustering performance on our datasets. We expanded the soft nearest neighbor loss by introducing an annealing temperature factor, which led to an even better disentanglement and k-Means clustering performance. We posit that our annealing mechanism helps by adapting the temperature needed throughout a training. Using our approach, we had a clustering accuracy of **95.5%** (96.2% on best run), **84.4%** (85.6% on best run), and **78.5%** (79.2% on best run) on the MNIST, Fashion-MNIST, and EMNIST Balanced datasets respectively, outperforming all our baseline models.

REFERENCES

- [1] Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.
- [2] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
- [3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
- [4] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
- [5] Xie, Junyuan, Ross Girshick, and Ali Farhadi. "Unsupervised deep embedding for clustering analysis." International conference on machine learning. 2016.
- [6] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

- [7] Jiang, Zhuxi, et al. "Variational deep embedding: An unsupervised and generative approach to clustering." arXiv preprint arXiv:1611.05148 (2016).
- [8] Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013).
- [9] Mukherjee, Sudipto, et al. "Clustergan: Latent space clustering in generative adversarial networks." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. 2019.
- [10] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.
- [11] Xiao, Han, Kashif Rasul, and Roland Vollgraf. "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." arXiv preprint arXiv:1708.07747 (2017).
- [12] McConville, Ryan, et al. "N2d: (not too) deep clustering via clustering the local manifold of an autoencoded embedding." arXiv preprint arXiv:1908.05968 (2019).
- [13] Tenenbaum, Joshua B., Vin De Silva, and John C. Langford. "A global geometric framework for nonlinear dimensionality reduction." science 290.5500 (2000): 2319-2323.
- [14] Maaten, Laurens van der, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of machine learning research 9. Nov (2008): 2579-2605.
- [15] McInnes, Leland, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction." arXiv preprint arXiv:1802.03426 (2018).
- [16] Salakhutdinov, Ruslan, and Geoff Hinton. "Learning a nonlinear embedding by preserving class neighbourhood structure." Artificial Intelligence and Statistics. 2007.
- [17] Frosst, Nicholas, Nicolas Papernot, and Geoffrey Hinton. "Analyzing and improving representations with the soft nearest neighbor loss." arXiv preprint arXiv:1902.01889 (2019).
- [18] Goldberger, Jacob, et al. "Neighbourhood components analysis." Advances in neural information processing systems. 2005.
- [19] Neelakantan, Arvind, et al. "Adding gradient noise improves learning for very deep networks." arXiv preprint arXiv:1511.06807 (2015).
- [20] Lloyd, Stuart. "Least squares quantization in PCM." IEEE transactions on information theory 28.2 (1982): 129-137.
- [21] He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.
- [22] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." Proceedings of the 27th international conference on machine learning (ICML-10). 2010.
- [23] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
- [24] Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics. 2007.
- [25] Yang, Yi, et al. "Image clustering using local discriminant models and global integration." IEEE Transactions on Image Processing 19.10 (2010): 2761-2773.
- [26] Chapelle, Olivier, Bernhard Scholkopf, and Alexander Zien. "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]." IEEE Transactions on Neural Networks 20.3 (2009): 542-542.
- [27] Hubert, Lawrence, and Phipps Arabie. "Comparing partitions." Journal of classification 2.1 (1985): 193-218.
- [28] Rousseeuw, Peter J. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis." Journal of computational and applied mathematics 20 (1987): 53-65.
- [29] Calinski, Tadeusz, and Jerzy Harabasz. "A dendrite method for cluster analysis." Communications in Statistics-theory and Methods 3.1 (1974): 1-27.
- [30] Davies, David L., and Donald W. Bouldin. "A cluster separation measure." IEEE transactions on pattern analysis and machine intelligence 2 (1979): 224-227.
- [31] Halkidi, Maria, Yannis Batistakis, and Michalis Vazirgiannis. "On clustering validation techniques." Journal of intelligent information systems 17.2-3 (2001): 107-145.
- [32] Cohen, Gregory, et al. "EMNIST: Extending MNIST to handwritten letters." 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017.
- [33] Bellman, Richard E. Adaptive control processes: a guided tour. Princeton university press, 2015.